Claudio De Sio Cesari

Il nuovo Java

Componenti fondamentali di un programma Java (NJ–002)

https://www.nuovojava.it

Componenti fondamentali di un programma Java

- Comprendere cosa significa sviluppare un programma in Java, cosa sono i processi e i paradigmi di programmazione (unità 2.1).
- Saper definire i concetti di classe, oggetto, variabile, metodo e costruttore (unità 2.1, 2.2).
- Saper dichiarare una classe (unità 2.2).
- Istanziare oggetti da una classe (unità 2.2).
- Utilizzare i membri pubblici di un oggetto sfruttando l'operatore dot (unità 2.2).
- Dichiarare ed invocare un metodo (unità 2.1, 2.2).
- Saper dichiarare e assegnare un valore ad una variabile (unità 2.1, 2.2).
- Saper definire ed utilizzare i diversi tipi di variabili (d'istanza, locali e parametri) (unità 2.2).
- Dichiarare ed invocare un metodo costruttore (unità 2.2).
- Saper definire un costruttore di default (unità 2.2).
- Saper definire una classe appartenente ad un package e saper importare classi da altri package (unità 2.3).
- Approfondimento online 2.1

Cosa significa creare un programma Java

- Ruolo della **main class** (**classe principale – classe del main**)
- Numero di classi di un programma
- Ruolo delle classi in un programma
- È fondamentale capire cosa significa **sviluppare software** (**software development**)

Software development (SD)

- Lo **sviluppo del software** è processo creativo che porta a creare un programma partendo da un'idea.
- Questo **processo** è composto da una sequenza di attività da svolgere, ma non esiste un processo standard che sia perfetto per ogni situazione di programmazione.
- Quindi, tali attività cambiano di tipologia, di ordine di esecuzione e di numero, a seconda del programma da creare

Il grande errore che commettono i programmatori poco esperti, è quello di pensare che per programmare basti saper programmare

SD: processo minimale

- Capire tutte le funzionalità che l'applicazione deve implementare (ovvero capire cosa deve fare il nostro programma)
- Trovare delle strategie per creare il programma nel modo migliore possibile (capire come implementare il nostro programma)
- Implementare con il codice le soluzioni ipotizzate (ovvero programmare)
- Testare il corretto funzionamento del nostro programma e risolvere eventuali problemi

Object Oriented Programming

- **OOP**: **programmazione orientata agli oggetti**, o più semplicemente **programmazione ad oggetti**
- Stile di programmazione basato su concetti ispirati dal mondo reale
- Paradigmi e principi OO

Paradigmi fondamentali

- **Incapsulamento**
- **Ereditarietà**
- **Polimorfismo**
- **Astrazione ***
- **Riuso**

Astrazione

- Potrebbe definirsi come:

«l'arte di sapersi concentrare solo sui dettagli veramente importanti di un'entità in un certo contesto»

- La usiamo ogni giorno in ogni istante della nostra vita:

«l'essere umano supera la complessità della realtà che lo circonda, grazie alla sua capacità di astrarsi»

Concetti alla base della sintassi Java

```
// dichiarazione di classe
public class SintassiBase {
    //dichiarazione del metodo main
    public static void main(String args[]) {
        // dichiarazione di variabili
        int variabile1 = 10;
        int variabile2 = 20;
        //stampa dell'addizione delle variabili
        System.out.println(variabile1 +
            variabile2);
    }
}
```

Concetti alla base della sintassi Java

- **Parole definite del linguaggio** (una volta si parlava di **parole chiave**)
- **Identificatori** (nomi)
- **Simboli definiti dal linguaggio** (quasi tutti **operatori** ma alcuni sono parole definite dal linguaggio)
- **Dati** (**valori** o **literals**)
- **Whitespace** (**spazi bianchi**)
- **Commenti** (meta-informazioni non utilizzate dal compilatore)

Concetti alla base della sintassi Java

- In particolare, una **riga di codice** può contenere una o più **istruzioni** (a volte solo una parte)
- Tipi di istruzioni
  - **Statement** (per esempio una **dichiarazione**)
  - **Espressione** (per esempio una **operazione**)

Variabili

- Permettono di memorizzare dati
- Possiamo gestire i dati definendo, modificando e per esempio stampando dati.
- È una sorta di contenitore in cui risiede un'informazione.
- Tale informazione, è il valore di un determinato tipo, per esempio un numero intero 1 di tipo *int*:

```
int numero = 1;
```

Variabili: dichiarazione

Per poter utilizzare una variabile, bisogna prima dichiararla, ovvero definirla. In particolare, le variabili vengono definite da:

- Una **lista di modificatori**. Tale lista è opzionale e i tipi di modificatori che si possono usare dipendono da vari fattori
- Un **tipo di dato**. Ogni variabile può contenere un certo tipo, per esempio un numero, un numero con la virgola o una lettera. Quando dichiariamo un tipo per una variabile, obblighiamo la variabile ad ospitare solo valori compatibile con il tipo
- Un **identificatore**. Esso serve per riferirsi alla variabile. Scegliere il nome più adatto per una variabile è fondamentale, è così che usiamo l'astrazione per le variabili, scegliendo un nome che ci risulti naturale e utile per poter interagire con il software
- Un **valore** (ovvero il dato contenuto nel contenitore). Esso deve essere compatibile con il tipo di dato usato per dichiarare la variabile. Durante l'esecuzione dell'applicazione può essere modificato. Una variabile senza un valore non ha senso

Variabili: dichiarazione

- Per dichiarazione di una variabile, intendiamo un'istruzione che segua la seguente sintassi (tra parentesi quadre è riportata la sintassi da considerare opzionale):

[modificatori] tipo_di_dato identificatore [= valore];

- Esempi:

```
int anni;
char lettera;
double tassoDiConversione;
int anni;
int mesi;
int anni, mesi;
```

Variabili: assegnazione

- Esempi:

```
anni = 16;
mesi = 2;
lettera = 'A';
tassoDiConversione = 1.1;
int anni;
anni = 16;
int anni = 16;
char variabileNonInizializzata;
System.out.println(variabileNonInizializzata);
```

Variabili: esempio

```
public class VariazioniDiVariabile {
    public static void main(String args[]) {
        double tassoDiConversione = 1.1;
        System.out.println("Valore attuale del tasso di conversione");
        System.out.println(tassoDiConversione);
        tassoDiConversione = 1.3;
        System.out.println("Valore attuale del tasso di conversione");
        System.out.println(tassoDiConversione);
        tassoDiConversione = tassoDiConversione - 0.3;
        System.out.println("Valore attuale del tasso di conversione");
        System.out.println(tassoDiConversione);
    }
}
```

Metodi

- Funzionalità definite da una sequenza di istruzioni
- Tutte le istruzioni che abbiamo eseguito con i nostri programmi erano contenute nel *main*
- È possibile anche dichiarare altri metodi con nomi e caratteristiche diverse
- Una volta definito un metodo, questo potrà essere invocato più volte. Invocare un metodo, significa eseguire la sequenza di istruzioni che definisce.
- Quindi, supponiamo ci serva eseguire più volte, in parti diverse del nostro codice, una certa sequenza di istruzioni, non sarà necessario riscriverle, ma bisognerà solamente invocare il metodo che le contiene nel momento in cui serve
- Concettualmente quindi, un metodo rappresenta una funzionalità, e di conseguenza il suo nome solitamente contiene un verbo

Metodi: dichiarazione

[modificatori] tipo_di_ritorno identificatore ([parametri]) {corpo_del_metodo}

- **Lista di modificatori**. Dipendono da vari fattori.
- **Tipo di ritorno**. Ogni metodo può ritornare un valore di un certo tipo. Se non ritorna valori (come nel caso del *main*), come tipo di ritorno viene usata la parola chiave di Java *void* (in inglese "vuoto"). Altrimenti bisogna specificare il tipo di questo valore (*int*, *double* etc.)
- **Identificatore**. È il nome del metodo e deve essere significativo;
- **Parametri** o **argomenti**. Una coppia di parentesi tonde con all'interno una lista di parametri (opzionale), ovvero dichiarazioni di variabili separate da virgole. Tali parametri possono contenere informazioni (valori) che possono essere passati al metodo quando esso viene invocato, e rappresentano quindi l'**input** del metodo
- **Corpo del metodo** (compreso tra parentesi graffe): insieme di istruzioni che verranno eseguite quando il metodo sarà invocato.

Metodi: esempio di refactoring

```
public class VariazioniDiVariabile {
    public static void main(String args[]) {
        double tassoDiConversione = 1.1;
        System.out.println("Valore attuale del tasso di conversione");
        System.out.println(tassoDiConversione);
        tassoDiConversione = 1.3;
        System.out.println("Valore attuale del tasso di conversione");
        System.out.println(tassoDiConversione);
        tassoDiConversione = tassoDiConversione - 0.3;
        System.out.println("Valore attuale del tasso di conversione");
        System.out.println(tassoDiConversione);
    }
}
```

Metodi: esempio di refactoring

```
public class VariazioniDiVariabile {
    public static void main(String args[]) {
        double tassoDiConversione = 1.1;
        stampaTassoDiConversione(tassoDiConversione);
        tassoDiConversione = tassoDiConversione + 0.2;
        stampaTassoDiConversione(tassoDiConversione);
        tassoDiConversione = tassoDiConversione - 0.3;
        stampaTassoDiConversione(tassoDiConversione);
    }
    public static void stampaTassoDiConversione(double tasso) {
        System.out.println("Valore attuale del tasso di conversione");
        System.out.println(tasso);
    }
}
```

Le basi della OOP

- Una **classe** è un'astrazione indicante un insieme di oggetti che condividono le stesse caratteristiche e le stesse funzionalità;
- Un **oggetto** è un'istanza (ovvero una creazione fisica) di una classe.

Esempio

```
public class Punto {
   public int x;
   public int y;
}
```

```
public class TestOggettiPunto {
   public static void main(String args[]) {
      Punto punto1;
      punto1 = new Punto();
      punto1.x = 2;
      punto1.y = 6;
      Punto punto2 = new Punto();
      punto2.x = 0;
      punto2.y = 1;
      System.out.println(punto1.x);
      System.out.println(punto1.y);
       System.out.println(punto2.x);
      System.out.println(punto2.y);
   }
}
```

Esempio (analisi)

```
public class Auto {
   public int numeroRuote = 4;
   public int cilindrata;
   public void accelera() {
      System.out.println("Sto accelerando con cilindrata:");
      System.out.println(cilindrata);
   }
}
```

```
public class TestOggettiAuto {
   public static void main(String args[]) {
      Auto fiat600 = new Auto();
      fiat600.cilindrata = 1100;
      fiat600.accelera();
      Auto california = new Auto();
      california.cilindrata = 4300;
      california.accelera();
   }
}
```

Astrazione

```
public class Termometro {
   public double temperatura;
   public void incrementaTemperatura(double i) {
      temperatura = temperatura + i;
   }
}


// REFACTORING
public class Termometro {
   public double temperatura;

   public void cambiaTemperatura(double differenza) {
      temperatura = temperatura + differenza;
   }
}
```

Astrazione

```
public class Termometro { // VERSIONE SINTETICA
   public double temp;

   public void cambiaTemp(double diff) {
      temp = temp + diff;
   }
}

public class Termometro { //VERSIONE LEGGIBILE
   public double temperatura;

   public void cambiaTemperatura(double differenza) {
      double nuovaTemperatura = temperatura + differenza;
      temperatura = nuovaTemperatura;
   }
}
```

Metodi: invocazione

```
public class GestioneTemperatura {
  public static void main(String args[]) {
    //Oggetto termometro in cucina:
    Termometro termometroCucina = new Termometro();
    termometroCucina.temperatura = 22.2;
    System.out.println("Temperatura attuale del termometro in cucina: ");
    System.out.println(termometroCucina.temperatura + " gradi");
    System.out.println("Ho aperto la finestra della cucina");
    termometroCucina.cambiaTemperatura(-2.5);
    System.out.println("Temperatura attuale del termometro in cucina: ");
    System.out.println(termometroCucina.temperatura + " gradi");
    //Oggetto termometro in soggiorno:
    Termometro termometroSoggiorno = new Termometro();
    termometroSoggiorno.temperatura = 18.6;
    System.out.println("Temperatura attuale del termometro in soggiorno: ")
    System.out.println(termometroSoggiorno.temperatura + " gradi");
    System.out.println("Ho acceso il riscaldamento in soggiorno");
    termometroSoggiorno.cambiaTemperatura(1.7);
    System.out.println("Temperatura attuale del termometro in soggiorno:");
    System.out.println(termometroSoggiorno.temperatura + " gradi");
  }
}
```

Metodi: altri tipi

```
public class Aritmetica {
  public int somma(int a, int b) {
    return (a + b);
  }
}
```

```
public class TestSomma{
  public static void main(String args[]) {
    Aritmetica oggettoAritmetica = new Aritmetica();
    int risultato = oggettoAritmetica.somma(5, 6);
    System.out.println(risultato);
  }
}
```

```
public class AritmeticaFissa {
    public int somma(){
        return (5 + 6);
    }
}
```

```
public class FabbricaPunti {
    public Punto dammiUnPunto() {
        return new Punto();
    }
}
```

```
public class Saluti {
  public void stampaSaluto() {
    System.out.println("Ciao");
  }
}
```

```
public class TestSaluti {
  public static void main(String args[]) {
    Saluti oggetto1 = new Saluti();
    oggetto1.stampaSaluto();
  }
}
```

Costanti: il modificatore *final*

```
public class Rettangolo {
    public final int NUMERO_LATI = 4;
    public int base;
    public int altezza;
}
```

Categorizzazione delle variabili

- Variabili d'istanza
- Variabili locali
  - Parametri formali

```
// USO DI VARIABILI D'ISTANZA
public class Test Rettangolo{
   public static void main(String args[]) {
      Rettangolo rettangolo = new Rettangolo();
      rettangolo.base = 10;
      rettangolo.altezza = 10;
      System.out.println(rettangolo.NUMERO_LATI);
   }
}
```

Variabili locali e parametri

```
public class Aritmetica {
   public int somma(int a, int b) {
      return (a + b);
   }
}
```

```
public class TestSomma{
   public static void main(String args[]) {
      Aritmetica oggettoAritmetica = new Aritmetica();
      int risultato = oggettoAritmetica.somma(5, 6);
      int a = 5, b = 6;
      risultato = oggettoAritmetica.somma(a, b);
      System.out.println(risultato);
   }
}
```

Costruttori

- Devono avere lo stesso nome della classe;
- Non dichiarano alcun tipo di ritorno;
- Sono invocati automaticamente (e solamente) ogni volta che è istanziato un oggetto, relativamente a quell'oggetto;
- Costruttore di default

```
public class Punto {
   public int x;
   public int y;
   public Punto(){
      System.out.println("Costruito un Punto!");
   }
}
```

Costruttori

```
public class Punto {
   public Punto() {
   }
   public Punto(int a, int b) {
      x = a;
      y = b;
   }
   public int x;
   public int y;
}
```

```
Punto p1 = new Punto();
Punto p2 = new Punto(12,14);
Punto p3 = new Punto();
p3.x = 12;
p3.y = 14;
```

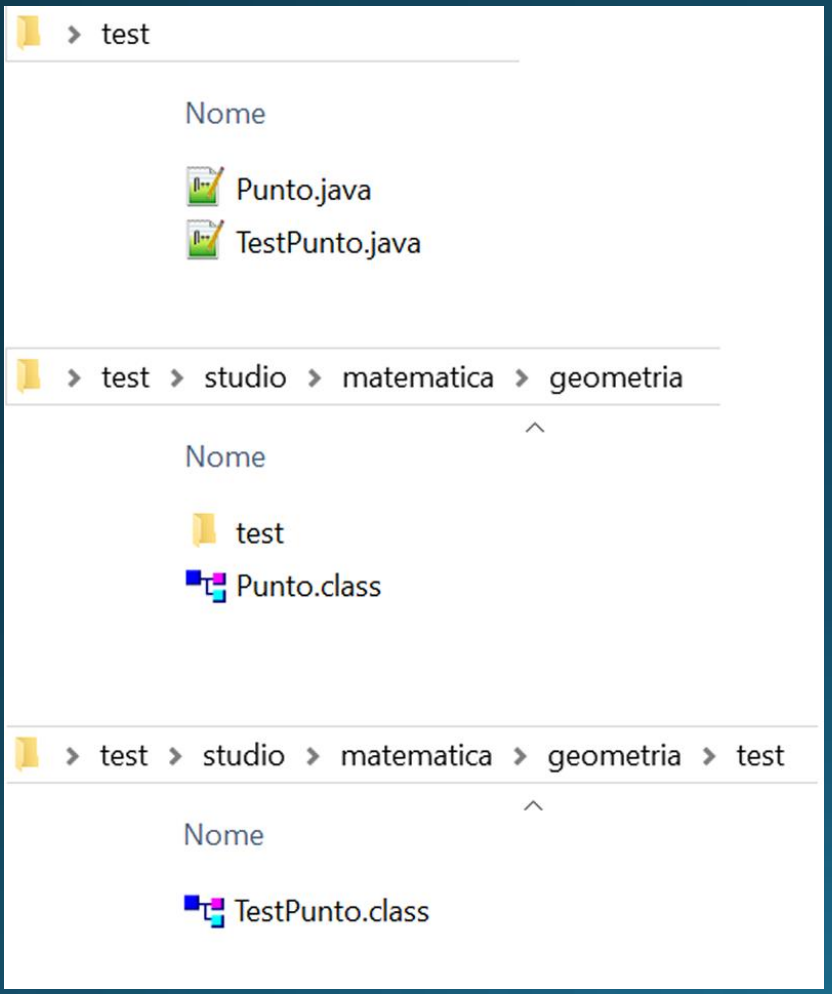
Package

- Fisicamente è una cartella, ma è dichiarata come *package* all'interno del codice sorgente
- Permette di raggruppare in un'unica entità, classi Java logicamente correlate
- Contiene i file .class
- Ha impatto sulla visibilità (scope) delle classi

```
package geometria;

public class Punto {
   public int x, y;
}
```

Package e import




```
package studio.matematica.geometria;

public class Punto {
   public int x, y;
}

package studio.matematica.geometria.test;

import studio.matematica.geometria.Punto;

public class TestPunto {
   public static void main (String args[])    {
      Punto p = new Punto();
      //altro codice omesso...
   }
}
```

Package: gestione manuale

- Complessa ed error-prone
- Necessità di IDE

javac -d . Punto.java TestPunto.java
java studio.matematica.geometria.test.TestPunto

Schema file sorgente

```
dichiarazione_di_package;

dichiarazione_di_import;

dichiarazione_di_classe {

    dichiarazioni_di_variabili;

    dichiarazioni_di_costruttori {
    }

    dichiarazioni_di_metodi{
    }
}
```

Sommario

- Comprendere cosa significa sviluppare un programma in Java, cosa sono i processi e i paradigmi di programmazione (unità 2.1).
- Saper definire i concetti di classe, oggetto, variabile, metodo e costruttore (unità 2.1, 2.2).
- Saper dichiarare una classe (unità 2.2).
- Istanziare oggetti da una classe (unità 2.2).
- Utilizzare i membri pubblici di un oggetto sfruttando l'operatore dot (unità 2.2).
- Dichiarare ed invocare un metodo (unità 2.1, 2.2).
- Saper dichiarare e assegnare un valore ad una variabile (unità 2.1, 2.2).
- Saper definire ed utilizzare i diversi tipi di variabili (d'istanza, locali e parametri) (unità 2.2).
- Dichiarare ed invocare un metodo costruttore (unità 2.2).
- Saper definire un costruttore di default (unità 2.2).
- Saper definire una classe appartenente ad un package e saper importare classi da altri package (unità 2.3).
- Approfondimento online 2.1